The SDSS-GriPhyN Cluster Finding Challenge Problem

James Annis[1], Steve Kent[1] Yong Zhao, Jens Voeckler, Ian Foster, Mike Wilde

[1]Experimental Astrophysics Group, Fermilab

Draft v1 November 20, 2001

We examine the Sloan Digital Sky Survey cluster finding algorithms as examples of virtual data. We describe one of these, the maxBcg algorithm that works on colors and luminosities. We then describe the superstructure necessary to turn this into a Challenge problem.

1 Introduction

The Sloan Digital Sky Survey[1] aims to map a quarter of the sky in five bandpasses relatively deeply into the galaxy population. A stated goal is object brightnesses and colors accurate to 2%. The combination of wide imaging and precise photometry makes it a near perfect tool to find clusters of galaxies.

1.1 Clusters of Galaxies

Clusters of galaxies are the largest bound structures in the universe; a good analogy is a hot gas cloud, where the molecules are galaxies. By counting clusters at a variety of redshifts as a function of mass, one is able to probe the evolution of structure in the universe. The number of the most massive clusters is a sensitive measure of the mass density Ω_m; combined with the cosmic microwave background measurements of the shape of the universe, these become a probe of the dark energy.

The traditional way to find clusters of galaxies is to look for galaxy overdensities. This technique has had a checkered history, but with the onset of the SDSS data it has proven to be very competitive as the precision photometry allows background and foreground galaxy rejection. There remain some fundamental limitations, the two most important of which that galaxies are a 1% constituent by mass of clusters and that there are a finite, and rather small, number of galaxies per cluster. The currently unique ability of the optical data to find clusters over a major fraction of the sky more than makes up for these limitations.

[1] For an overview, see
http://www.sdss.org
For details, see both
http://www.astro.princeton.edu/PBOOK/welcome.htm and
http://xxx.lanl.gov/abs/astro-ph/0006396
For access to data and playing around, see
http://skyserver.fnal.gov/en/

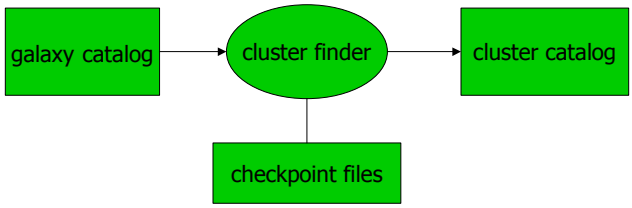
Figure 1: A rich intermediate redshift cluster of galaxies in the SDSS data. You are to be looking at the red extended galaxy at the upper right side of image center. If you have problems seeing the cluster, which is a rich cluster, then you have noticed the problem of low contrast over the background that has plagued cluster catalog making in the past. Color helps, to the point of bing revolutionary; the blue objects in lower right side of center are certainly foreground galaxies.

1.2 Cluster Finding Algorithms as Virtual Data Engines

The concept of virtual data breaks into two parts: transparency with respect to location and transparency with respect to existence. The latter, derived data generated on the fly, is the research we wish to explore. Cluster catalogs are nearly perfect examples of derived data. Currently they are produced in production mode, where one creates a catalog against all existing data. The codes may be rearranged to human driven mode, when the data are generated on demand. The latter has significant benefits, as the parameters desired by the scientist may not have been those that were used in the production mode. Furthermore, the intermediate products of cluster finding are interesting derived data in their own right.

Virtual Data Engine:

-Galaxy catalog is input
-Algorithm runs
    -creating intermediate files
-Cluster catalog is output.



Furthermore, there is no one correct way to select clusters of galaxies[2]. Nature has made a clean break at the scale of galaxies: galaxies and entities smaller are cleanly identifiable as single entities; above galaxies the entities are statistical. Given that, there are many different ways to identify clusters of galaxies. The SDSS currently is exploring 6 different cluster catalogs.

Each of these catalogs are derived data sets. It is worth pointing out that

a. Each algorithms have changeable parameters,
b. Each algorithm evolves and hence has version numbers,
c. The underlying data can change as the reduction or calibration is re-performed.

We thus point out that versioning and the associated bookkeeping is important.

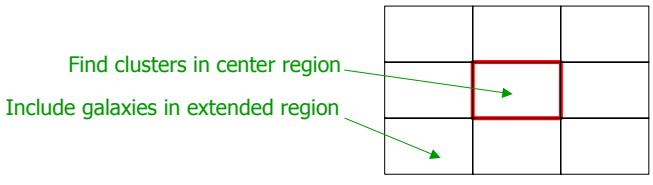
1.3 Problem Analysis

The basic procedure to find clusters is to count the number of galaxies within some range about a given galaxy. This is an N^2 process[3]. Note that the procedure is done for each galaxy in the catalog. The problem is thus computationally expensive, but it is in fact balanced with I/O requirements; with the appropriate choices of parameters it can be made either an I/O bound problem or a CPU bound problem.

Table 1: Computational and Storage Requirements for the full SDSS

Area	7000 sq-degrees
Storage	1540 Gigabytes
Compute[4]	7000 CPU-hours (500 Mhz PIII, 1 Gig RAM)

Table 1 shows the scale of the problem. The problem can be made embarrassingly parallel by setting an upper limit to the angular size of the clusters. This is in effect setting a lower limit to the distance to the cluster. Then one works on a central region, with a buffer zone around that region of angular size of the upper limit. One only locates clusters in the central region, but uses the information from the buffer zone for the calculation. The buffer zone has, in a purely distributed computing environment, a cost as it is replicated data; this is not a strong constraint. A natural size for the central region is about 7 square degrees; this translates to 7 hours of compute time and 1.5 Gig that must be transferred. The latter is an upper limit; the first step in the cluster finding algorithm extracts from the full data set the galaxies, and only those measurements on the galaxies of interest and then produces new files containing this data. The new files mass about a factor of 40



[2] Except, of course, that my algorithm is clearly superior.
[3] Though with us the use of metadata stored on trees it can be brought down to a N log(N) problem
[4] For the MaxBcg algorithm, the clearly superior one mentioned earlier.

smaller than the full data set, and thus after the first stage, one is transferring about 35 Meg around for the 7 sq-degree region.

Table 2: Computational and Storage Requirements for the Natural Unit

Area	7 sq-degrees
Storage	1.5 Gigabytes
Compute[5]	7 CPU-hours (500 Mhz PIII, 1 Gig RAM)

The work proceeds through many stages and through many intermediate files that can be used as a form of checkpoint. This is an advantage.

To summarize, cluster finding is a good choice for the initial challenge problem as

1. cluster catalogs are a good example of derived data,
2. cluster catalog creation is roughly compute and storage balanced (~5 CPU-hours/Gig).

2 The Cluster Catalog Generation

The algorithm that we will use is called MaxBcg, from "MAXimum likelihood determination of the Brightest Cluster Galaxy". It has had considerable success on the SDSS dataset, for which it was designed. It is an astrophysically based method, in that it incorporates much knowledge about the object under study rather than performing a purely statistical search. Examples of the latter would be looking for overdensities in a 7 dimensional space of position and fluxes., or by searching for compact cells in a Voronoi tessellation of space broken up into a large number of color slices. One of other benefits of the maxBcg algorithm is that it is fast compared with the other algorithms.

2.1 The MaxBCG Algorithm

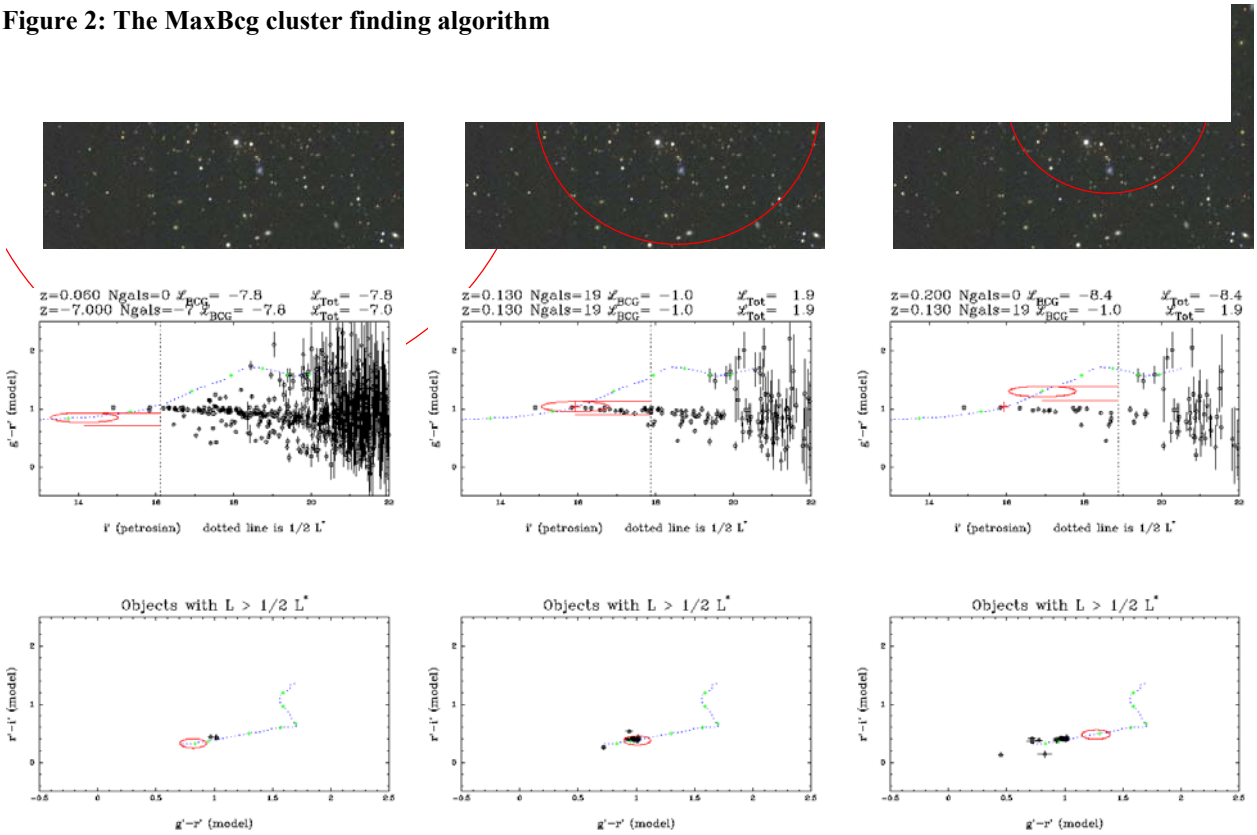
At its most abstract, the algorithm moves a cylinder around in a 5 dimensional space, calculating the likelihood at each point. Doing so on a grid in 5 dimensions would be prohibitively expensive, not to mention silly. There is a natural, adaptive, non-linear grid that is optimal to use, and gets its structure from the domain knowledge about the space. The 5-dimensional space is that of:

1. two spatial dimensions, Right Ascension (RA) and Declination (dec)[6]
2. two color dimensions, g-r and r-i[7]
3. one brightness dimension, i

The natural grid in the two spatial dimensions is that of the galaxies themselves; calculations are performed at the location of a galaxy. The two color and one brightness dimension would still be

[5] For the MaxBcg algorithm, the clearly superior one mentioned earlier.
[6] RA and Dec make sense for observing the fixed stars from a spinning, revolving platform, but the details are unnecessary here; think of them as something like azimuth and elevation.
[7] g,r,i denote energy fluxes through bandpasses corresponding to green, red, and beyond red light. The flux is in log, so the g-r notation implies a flux ratio. There is a –2.5 multiplicative factor present for historical reasons; unfortunately the French Revolution did not exert its influence on astronomy.

Figure 2: The MaxBcg cluster finding algorithm



prohibitively expensive, except that there is a one-dimensional path through those three dimensions that the vast majority of cluster galaxies follow. That track is parameterized by the redshift, z, and the track itself may be calibrated by a combination of galaxy formation/spectral synthesis models and the data themselves. The blue dotted line in figure 2 shows the projections of this 1-d track onto the two most useful projections, g-r vs. i and r-i vs. g-r. The red lines show the acceptence window: on the images they show a radius of 0.5 Megaparsec (Mpc; this is half of the radius usually used) and objects inside that radius show up in the plots; on the bottom plot the ellipse shows the range of colors allowed about the projection point; and in the middle plot the horizontal lines shows the acceptance window in color and brightness. The ellipse in that figure shows something else: the mean and 2-σ dispersion about the mean of the properties of the brightest galaxy in each cluster of galaxies, for a large sample of clusters.

The algorithm works like this: for each galaxy, loop over all redshifts. At a given redshift, calculate the likelihood that the observed properties of this galaxy match the known properties of brightest cluster galaxies as they would be seen at that redshift. Weight this likelihood by the log of the number of galaxies that are inside the acceptance window, again at that redshift. Having made this calculation over all redshifts, choose the redshift which maximizes the probability for this galaxy and record the properties, probability, and redshift[8]. Having done this for all galaxies, once again walk over all galaxies and for each galaxy ask: does this galaxy have the highest probability of all

[8] This photometric redshift turns out to be a very good estimate of the cluster redshift, good enough that a spectroscopic redshift of the cluster is superfluous.

galaxies within 1 Mpc and 0.05 of its redshift, at its redshift? If so, there is a cluster present, and this galaxy is the brightest cluster galaxy and by definition the center of the cluster. If not, move on.

Finally, there is the step of walking through each galaxy and extracting those that have been denoted as the centers of clusters. This is the final cluster catalog creation step.

2.2 Transformations

We can describe the algorithm as a series of transformations, each of which takes input, makes a computation, and creates new output files. The code of the algorithm is already organized in this fashion, the challenge is to use the Virtual Data Language to describe both the transformations and the files.The series of transformations is shown in table 2.

Table 3: The maxBcg transformations

fieldPrep:	Create field files from the galaxy catalog.	
	Why:	Reduce data volume, increase I/O speed by large factors
	Input:	tsObj files
	Output:	field files
brgSearch:	Calculate the unweighted BCG likelihood for each galaxy	
	Why:	The unweighted likelihood may be used to filter out unlikely candidates
	Input:	field files
	Output:	brg files
bcgSearch:	Calculate the weighted BCG likelihood for each galaxy.	
	Why:	The heart of the algorithm, and expensive.
	Input:	brg files
	Output:	core files
bcgCoalesce:	Is this galaxy the most likely galaxy in the neighborhood?	
	Why:	Find peaks in the probability, for they are clusters
	Input:	core files
	Output:	cluster files
getCatalog:	Find peaks in the probability, for they are clusters	
	Why:	Remove extraneous data
	Input:	cluster files
	Output:	cluster catalog

For the entire cluster finding code, there are two parameter files. One contains parameters that control the code, the other a lookup table of the expected colors and magnitudes of various galaxies at various redshifts (called the k-correction, for reasons lost in the mists of time).

3 The Cluster Finding Challenge

The Cluster finding challenge problem aims to exercise the use of derived data catalogs and metadata, replica catalogs, transformation catalogs including DAG creation, and code migration. The last of these is a logically distinct effort, but incorporates nicely here.

3.1 Description

Aim: Demonstrate a catalog-driven data derivation tool that, when given a request for data, determines whether computation is required to generate it, and if so schedules it.

Plan: Make a simple demo: given a display of the sky with areas highlighted in three ways (Clusters found, data exists, and no data) allow the user to make a plot of the clusters in the area. This is done by:

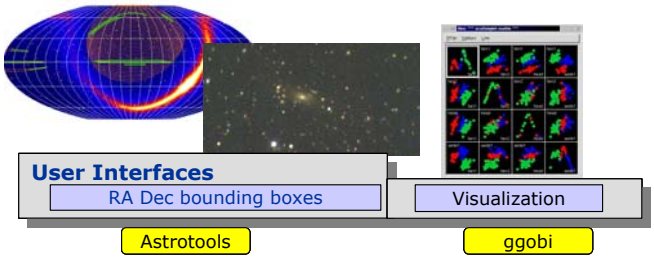
1. Drawing a box on the sky image
2. Computing a bounding box
3. Creating sub-bounding boxes where cluster catalogs don't exist
4. Finding those sub-bounding boxes where galaxy data does exist
5. Using the transformation catalog to compute the DAG necessary to calculate the cluster catalog in those sub-bounding boxes
6. Managing the computation
7. Updating the cluster catalog
8. Sending the cluster catalog to the user interface

Components: There are four distinct components to this system:

1. The User Interface (Fermilab)
2. The Virtual Data Catalog (UC)
3. The Derived Data Catalog (Fermilab)
4. The Computation Manager (UW)

3.2 User Interface *(Fermilab)*

The user interface presents a sky image to the user. Upon selection of a region, the user interface computes the RA and Dec bounding box of the region and a SQL request is constructed. The SQL request is treated as a URL, and the visualization tool is invoked aimed at that URL.



The technologies involved:

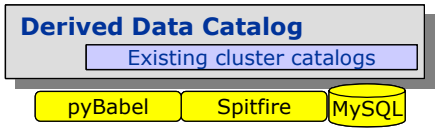
astrotools: The SDSS pipeline codes are generically called astrotools[9]. Astrotools has the coordinate transform routines that are necessary for the user interface.

ggobi: The AT&T multidimensional visualization package xgobi has evolved into a Gnome front ended version, ggobi[10]. It also has hook that allows one to access XML data directly from the web, as in ggobi -x http://localhost:8000/sqldb/demo/query.xsql?query=select from cluster where ngals>=20 . Here one is finding all clusters above a given richness, without an explicit bounding box. The main thrust of ggobi is rapid exploration of many dimensional data.

It is worth noting that it is this request drives the machinery

3.3 The Derived Data Catalog *(Fermilab)*

The derived data catalog (DDC) holds the materialized data, here the cluster catalog. It accepts RA-Dec SQL queries over the web and return XML encoded data. The DDC consists of a database and a pair of web servers, the second being an interceptor layer that translates the XML into the XML format needed down stream, in our case the ggobi format.



The technologies involved:

pyBabel: The interceptor layer is a python web server that takes in the SQL queries, retransmits them to the Spitfire web server, catches the XML response from Spitfire, translates it for ggobi, and sends it on. It sits on port 8000, as in ggobi -x http://localhost:8000/ and redirects the query to port 8080, where Spitfire lives.

Spitfire: Spitfire[11] is a grid enabled relational database. More exactly, it is a set of grid enabled Java servlets running in the Apache Tomcat Java web server, and which acts as a JDBC layer on many relational databases. It allows SQL queries, table updates, and deletions to come in over the web, and the results to go back in XML format.

MySQL: The base database for the DDC is MySQL, as that is what spitfire comes configured to run right out of the box.

At this point we have a functioning system, albeit one without the ability to generate virtual data.

---

[9] http://www-sdss.fnal.gov:8000/sdss.pipe.html
[10] http://www.ggobi.org
[11] http://hep-proj-spitfire.web.cern.ch/hep-proj-spitfire/share/spitfire/doc/index.html

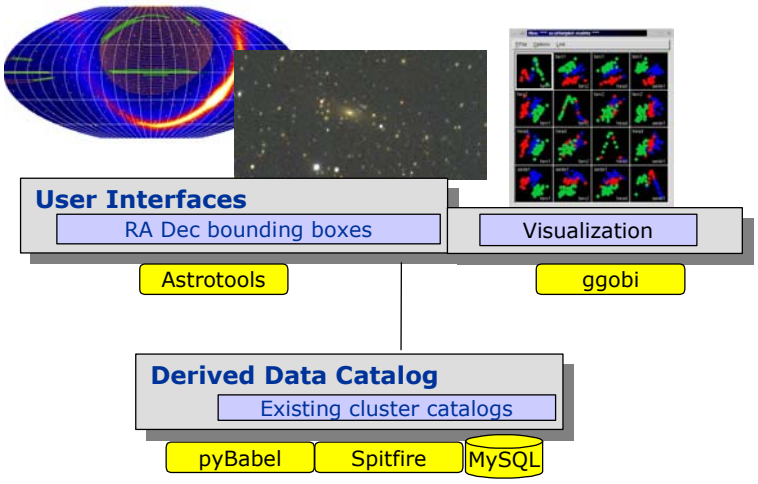


Figure 3: The concrete data system.

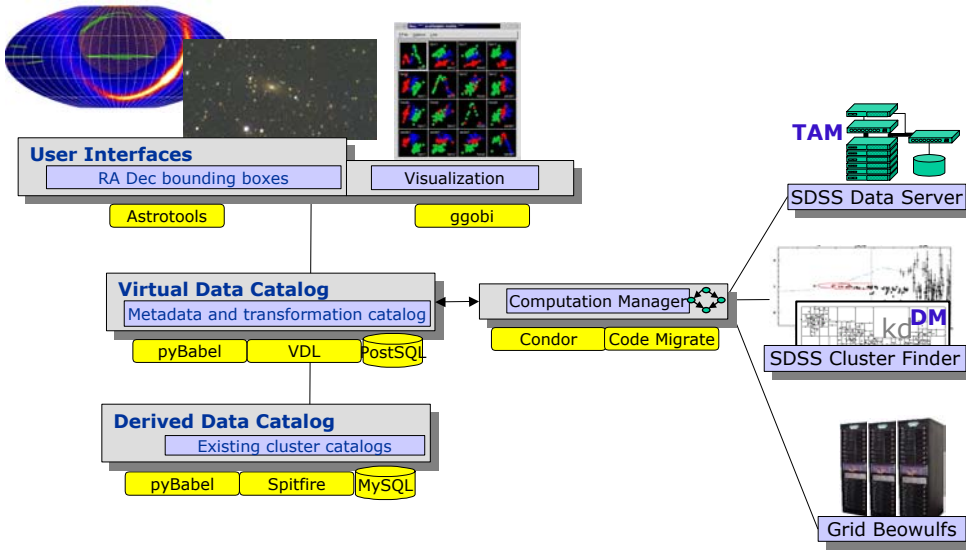
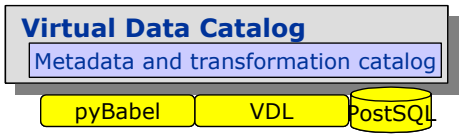


Figure 4: The full virtual data system

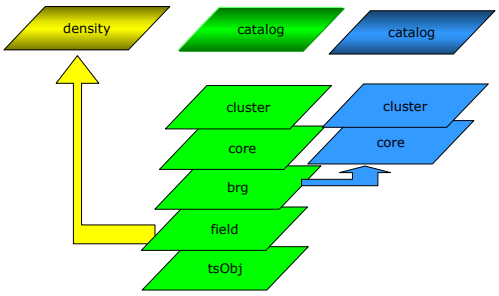
3.4 Virtual Data Catalog (UC/Fermilab)

The VDC is an interceptor layer of a more formidable level. The user interface and the derived data catalog together construct a concrete data system, seen in figure 3. When the virtual data catalog is placed as an interceptor in front of the derived data catalog, it increases the capability of the system by allowing both the materialization of new cluster catalogs and keeping track of what was done to produce them: both the computation and the bookkeeping.

The VDC takes in the request for a URL page, splits out the SQL request, and from that the bounding box. It then splits the bounding box up into bounding boxes for which there exist cluster catalogs, and into bounding boxes for which there exist galaxy data and hence the capability to generate cluster catalogs. If the latter is an non-empty set, the VDC calculates the DAG and sends it on to the Computation Manager. When the computation manager finishes, there are new data to be placed into the derived data catalog, and new metadata and bookkeeping data to be placed in the VDC. Finally, the URL request is honored.



There is an interesting mechanism here: the VDC's conception of the cluster catalog data in the derived data catalog is that of a series of transformations (see table 3.) The top layer, the cluster catalog, only knows that it was created from cluster files, not that ultimately it came from tsObj files. The VDC should recurse the transformations back to a place where it either it knows where to find the files or the files don't exist. This allows one a) to rerun the cluster finder with different parameters, skipping some of the more time intensive steps (the blue branch), and more interestingly b) to run other analyses that depend on prior files (the yellow branch). The latter is the beginning of a plug and play approach to virtual data, in the sense that new algorithms need only know their input, not necessarily the entire transformation tree.

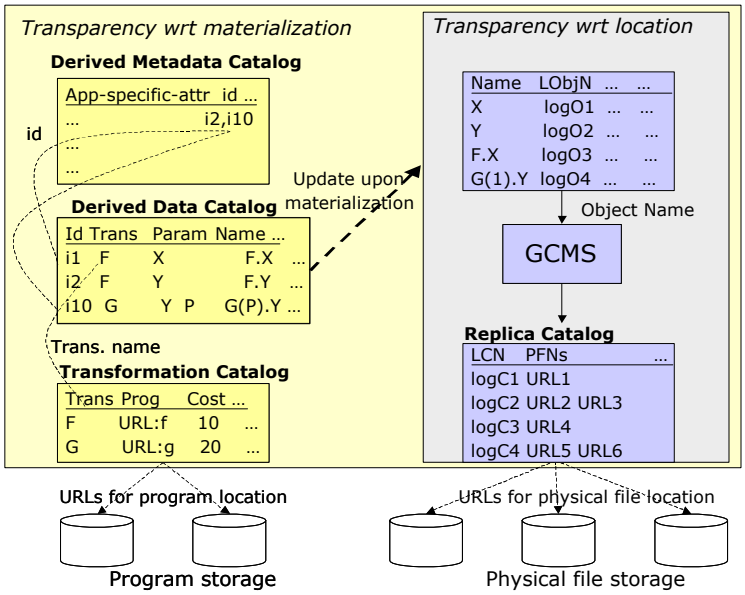


The technologies involved:

PyBabel: a web interface to the VDC.
Virtual Data Catalog: the code without a name
PostgreSQL: the database that the code without a name uses.

3.4.1 The VDC Transformation Catalog

The first transformation, fieldPrep, is the place to start.



Rc parameters.par /daedaleos/parameters/parameter7.par
Rc kcorr.par /daedaleos/evo/kcorr.par

Begin astrotools
Arg –f fieldPrep
Stdout fieldPrep.log
File i tsObj-
File o field-
End
Rc fieldPrep /data/run10/fieldprep.tcl
Rc fieldPrep.log /data/run10/fieldprep.log
Rc field- /data/run10/fields/field-%s-%s-%s.par
Rc tsObj- /data/tsObj/%s/%s/tsObj-%s-%s-%s-%s.par

Ok, what I mean by the last two are files that look like:
Field-752-3-123.par
Field-752-4-433.par
and so on; at run time

3.4.2 Simple Pseudo-Code **(Is this code close to this, in reality?)**

For the VDC:

overlap(bounding-box, &list-of-existing-bounding-boxes, &list-of-nonexisting-bounding-boxes): return the list of existing and new BBs needed to construct new BB

add_region(bounding-box, list-of-clusters): adds BB to the VDC, and also adds the clusters to the cluster catalog

delete_region(bounding-box): delete BB from the VDC, delete associated clusters from the cluster catalog.

To simplify: Bounding boxes cannot overlap; can create, but can't delete arbitrary bounding boxes.

build_task_list(entire-bounding-box, ntasks, &list-of-task-bounding-boxes)

build_task(bounding-box, program, parameter-file, &task-description)

get_files(bounding-box, &list-of-tsobj)

```
build_dag(entire-bounding-box, ntasks, program, parameter-file, &dag) {
   init_dag(dag);
   build_task_list(entire-bounding-box, ntasks, &list-of-task-bounding-boxes);
   for each bounding-box in list-of-task-bounding-boxes {
      build_task_dag(bounding-box, program, parameter-file, &task-description);
      add_to_dag(dag, task-description);
   }
}

build_task_dag(bounding-box, program, parameter-file, &task-description) {
    get_files(bounding-box, &list-of-tsobj);
    build_task_description(list-of-tsobj, program, parameter-file, &task-description);
}
```

build_task_list and get_files call Astrotools code.

The build_task_dag might want to be smart about scheduling data movements, etc.

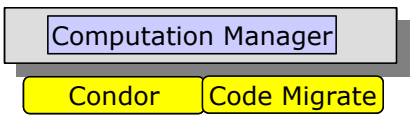
Simplification: no wraparounds (?).

3.5 Computation Manager (UW/Fermilab)

The Computation Manager allows us to connect to significant grid resources. This is essentially Condor-g, and hence the problem is just to incorporate it. There is a significant new capability to be built, that of a more powerful code migration.

The technologies involved:

Condor-g: Condor[12] manages computations on distributed resources.
Code Migrator: new code to handle the migration of astrotools and related scripting language based science codes.



[12] http://www.cs.wisc.edu/condor/

3.5.1 Code Migration Subsystem

The code migration problem is essentially that of using Astrotools in a Condor environment without having a shared code serving file system.

The problem has two parts. The first is the dynamic libraries that TCL access. The second is the way TCL sources scripts as part of startup. The solution is the Fermilab UPS environment[13].

Astrotools is the SDSS software package. Is is a collection of C extensions to TCL, arranged in a layered fashion over a set of circa 10 public domain software packages. Note that we are not using TCL as a shell substitute; the packages are linked at compile time into a TCL interpreter, into a executable named astrotools.

The Fermilab UPS package performs duties much like RedHat's rpm, except less conveniently. It does, however, have one very large capability rpm lacks: Users can switch between two different software packages that require the same underpinning, but with a different version, without reinstalling software.

When one invokes ups, ("setup astrotools") one is asking it to query a flat file database (always on a locally accessible disk, usually specific for the local machine) for what environmental variables need to be set (e.g., ASTROTOOLS_DIR for the astrotools source and libraries), what path needs to be added to PATH, and what dependent products exist. Dependent products are then set up using ups in a recursive fashion. (Dependent products for astrotools include the SAOimage image display program, and the PGPLOT plotting package.)

Now, TCL/Tk allow one to have startup TCL startup scripts, to be run as the executable starts. These are handled in UPS via environmental variables such as ASTROTOOLS_STARTUP. All of the TCL-based packages have such scripts, underpinning or not[14].

Now, what prevents us from running Astrotools via Condor except on machines with the SDSS environment installed, is the variety of scripts and libraries that are required at startup that the user

---

[13] UPS/UPD is the software product version and distribution support infrastructure at Fermilab. It is a set of C and Perl scripts. The database is a set of ASCII flat files pointed to by environment variables. The Fermilab ODS department wants to do some work over the next year to bring more into alignment ups/upd and rpm/autorpm/systracker - which is a layer on top of autormp using CVS as a repository. The other differences between UPS/UPD and RPM that might be relevant are

a) ups does not require root access to install software
b) ups supports the concept of "flavor" and "qualifiers" that provide for the support of different OS and OS environment versions and types. Also, support for node specific s/w startup and configuration.
c) ups supports startup, shutdown, tailor, configure etc.
d) as mentioned below ups supports specification of dependencies between s/w packages and upd allows automatic distribution of dependent packages.

For more on UPS, see http://www.fnal.gov/docs/products/ups/. For installing UPS/UPD, see http://www.fnal.gov/docs/products/bootstrap/ and in particular ftp://ftp.fnal.gov/products/bootstrap/v2_0/index.html)

[14] Startup is the act of starting the TCL interpreter, and that is the same as starting any Unix program. Applications are large scripts. These are usually run from inside TCL, though there is a facility to do it from the command line:

```
unix% astrotools -f my-large-script.tcl &
```

has no control over. The ones that I am sure of are the tcl startup scripts, and a library that TCL wants. Excluding those, could one ship around the binary of astrotools to a random machine and have it work? (For example, could gifs of plots be made in batch mode because pgplot is linked in to the astrotools code?) The answer is yes.

But getting the TCL startup scripts and the TCL library out to the Condor pool is a real problem. Solving it may make the exploring of the last paragraph unnecessary, as the easiest way to proceed is to either a) use the UPS database, or b) use before and after environment snapshots to find out what underpinnings are needed, taring them all up, and shipping the tarball around to the condor pool.

Note that I am not talking about a user's science code trying to source a random file during program execution. That too could be solved this way, but there are other ways of dealing with that. What I need help solving is the problem of custom run time libraries and of TCL initialization scripts in a Condor distributed computing environment.

LIGO has the same problem, by the way. They are handling it the same way that I am handling it for my science: the Condor pool is restricted to only those machines that have the full environment installed.

How can we get a list of what the application needs in order to run? The flat files of the UPS database have all the necessary information for the startup scripts.

Table 4 shows the UPS database for astrotools. Note that it explicitly calls out the startup script astrotoolsStartup.tcl. (fslalib and sdssmath are both dependent products. Somewhere below sdssmath in the dependency tree lies Tk, and below that, TCL.) Table 5 shows the UPS database file for TCL, and table 6 shows the setup.sh file for TCL. What is a "setup.sh" file? Part of the UPS system that lives with the product. All of the codes we talk about are products, in the sense that they have directory structures that look like:

```
[annis@espresso v7_4dfa]$ pwd
/p/tcl/v7_4dfa
[annis@espresso v7_4dfa]$ ls
bin  BUILD_INFO  include  lib  Makefile  man  src  tcl  tcl7.4  ups  ups-old
[annis@espresso v7_4dfa]$
```

They are not spread out across system directories; they care compact. The ups directory above contains "setup.sh" and "setup.csh" files that are run as part of doing the "setup tcl" command.

Table 4: UPS DB file for Astrotools

```
FILE = TABLE
PRODUCT = astrotools

Group:
Flavor=Linux+2.2
```

```
Qualifiers=""

Common:
  Action=setup
    setupRequired("-f ${UPS_PROD_FLAVOR} fslalib v2_1_2a")
    setupRequired("-f ${UPS_PROD_FLAVOR} sdssmath v5_0")
    proddir()
    setupenv()
    envSet(ASTROTOOLS_STARTUP, ${UPS_PROD_DIR}/etc/astrotoolsStartup.tcl)
    pathAppend(PATH, ${UPS_PROD_DIR}/bin)
End:
```

Table 5: UPS DB file for TCL

```
File=Table
Product=tcl
#***********************************************
# Starting Group definition
Group:

Flavor=Linux+2.2
Qualifiers=""

Common:
   Action=setup
     proddir()
     setupenv()
     sourceRequired(${UPS_UPS_DIR}/setup.${UPS_SHELL},UPS_ENV)
End:
# End Group definition
#***********************************************
```

Table 6: setup.sh file for TCL

```
PATH="${TCL_DIR}/bin:$PATH"
TCL_LIBRARY=$TCL_DIR/lib/tcl7.4
export TCL_LIBRARY PATH
```

The way to approach the code migration is to:

1. Read the UPS db file for the science code

2. Look for the directives “setupRequired”, “sourceRequired”, and “envSet”
3. If envSet:
    a. Check to see if it the *_STARTUP environmental variable; if so, then the product must be migrated
4. If sourceRequired:
    a. Check to see if it is a setup.sh or setup.csh file; if so, read it
    b. Look for environmental variables containing LIBRARY; if present, the product must be migrated
5. If setupRequired, recurse.

At the end one will know which products must be migrated, and which environmental variables must be set. I believe that those two conditions are the ones that are imply migration, and without them the necessary parts of the product have been linked into the astrotools binary and migration is unnecessary.

Success may be tested by editing a file named “cmd” and adding
`echo [stripeLimits 10]`
to it. The test is to run successfully over a Condor pool the command:
`astrotools –f cmd`

There is also the discussion of whether you send the tar with the job, or whether you "prep" the Condor pool and leave the code in place for a week. The latter would require developing a mechanism by which Condor makes sure that the job feels "at home" on the remote machine. We can do it every time we run a job, we can do it once for a "cluster of jobs", we can do it "lazy" only when we need it, or .... This is a very important problem that we need to solve.

4 Task List

- Yong:
    - Get tsObj file from Jim
    - Do a fieldPrep by hand
    - Create a transformation catalog entry for fieldPrep
    - Do a fieldPrep via VDL created DAG
    - Move on to the other transformations
- Alain:
    - Code migration, test `astrotools –f cmd`
- Jens:
    - VDL extension as necessary
- Jim:
    - Complete user interface
    - Set up CVS repository for VDL
    - Set up CVS repository for pyBabel and other codes